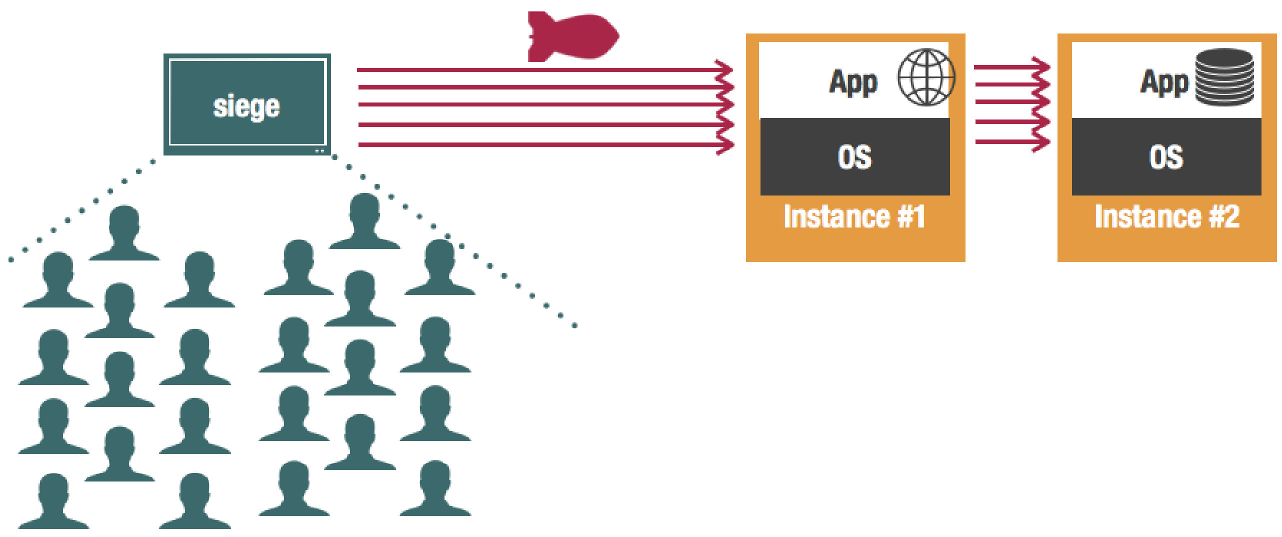
Table of Contents



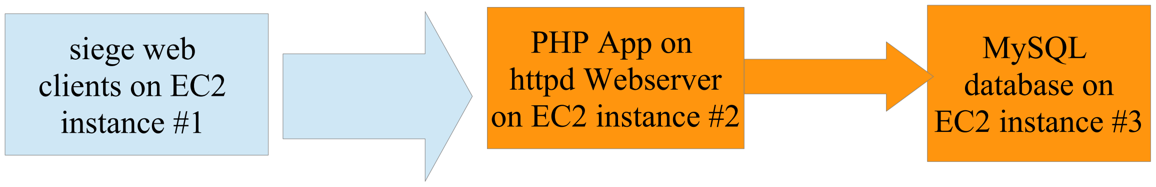
I. Prepare before Activity

II. Activity 2 Deploy IaaS Web Application

Run php web app on EC2.

I. Prepare before Activity

Goal: In the same AZ


siege web clients on EC2 instance #1
PHP App on httpd Webserver on EC2 instance #2
MySQL database on EC2 instance #3

Set up "siege" your web client for load testing to run on your Amazon EC2 instance (Amazon Linux)

Get an EC2 Amazon Linux 2023 instance (t2.micro or t3.micro instance free tier). Select an AZ, and note which one you use.

SSH into your instance using your .pem file

Install gcc

```
$ sudo yum install gcc
```

Press 'y' for each of the prompts that shows up. Note that you're logged in as ec2-user, so you need to *sudo* all of these commands.

Download, build, and install siege on your EC2 instance from

https://github.com/JoeDog/siege/

Read the README and INSTALL to get the full instructions on what you need to do (i.e., install prerequisites, generate the config file, run configure, make, make install).

After installation, run siege against a web application

```
$ siege -c5 -d0.5 -r10 http://www.google.com
```

Understand siege options (these are just some selected examples)

```
$ man siege
```

```
       -c NUM, --concurrent=NUM
           This option allows you to set the concurrent number of users. The
total number of users is technically limited to your computer's resources.
```

```
           You should not configure more users than your web server is
configured to handle. For example, the default apache configuration is capped at
255 threads. If you run siege with -c 1024, then 769 siege users are left
waiting for an apache handler.

           For this reason, the default siege configuration is capped at 255
users.  You can increase that number inside siege.conf but if you make a mess,
then please don't complain to us.
```

-d NUM, --delay=NUM

```
           This option instructs siege how long to delay between each page
request.  The value NUM represents the number of seconds between each one. This
number can be a decimal value. In fact the default is half a second (--
delay=0.5).

           The time between delay requests is NOT applied toward the transaction
time. If two 0.1 second transactions have a 2 second delay between them, their
average transaction time is run is 0.1 seconds. It is applied toward the total
elapsed time. In this scenario, the elapsed time would be 2.2 seconds.

           NOTE: when the parser is enabled (see: -p/--parser), there is no
delay between the page and its elements, i.e., style sheets, javascripts, etc.
The delay is only between page requests.
```

Look at the output from siege. Understand what the output means.

```
HTTP/1.1 200     7.33 secs:    2556 bytes ==> GET  /
HTTP/1.1 200     0.00 secs:    3490 bytes ==> GET  /styles.css
HTTP/1.1 200     0.00 secs:    3490 bytes ==> GET  /styles.css
HTTP/1.1 200     0.01 secs:     251 bytes ==> GET  /css?family=Lobster+Two
HTTP/1.1 200     0.01 secs:     251 bytes ==> GET  /css?family=Lobster+Two

Transactions:                    30 hits
Availability:                100.00 %
Elapsed time:                  8.34 secs
Data transferred:              0.06 MB
Response time:                     2.44 secs
Transaction rate:              3.60 trans/sec
Throughput:                    0.01 MB/sec
Concurrency:                   8.78
Successful transactions:             30
Failed transactions:                  0
Longest transaction:               7.33
Shortest transaction:              0.01
```

II. Activity 2 Deploy IaaS Web Application

A. Set up MariaDB database

Get an EC2 Amazon Linux 2 instance (micro instance free tier). Use the same AZ as your previous instance.

SSH into your instance using your .pem file

Install MariaDB

```
$ sudo yum install mariadb105-server
```

Press ‘y’ for each of the prompts that shows up. Note that you’re logged in as ec2-user, so you need to *sudo* all of these commands.

Configure MariaDB server to start up automatically on reboot.

```
$ sudo systemctl enable mariadb
```

Start the MariaDB server

```
$ sudo systemctl start mariadb
```

You can verify thatMariaDB is on by running the following command:

```
$ sudo systemctl is-enabled mariadb
```

Change the MariaDB root password. Use your own password. Remember it. This is not the most secure configuration, but it will do for now.

```
$ mysqladmin -u root password 'new-password'
```

Connection to your MariaDB server. You can do this using the command line client on your EC2 instance using the command:

```
$ mysql -uroot -p
```

Create a database called "ebdb".

Create a table called user with no data inside, containing 2 fields: id and username.

user

id (int)	username (varchar 64)

B. Prepare php web application

Download all the web app files for activity 2 from myCourseville onto your notebook.

There are 3 files: index.php, styles.css, logo_aws_reduced.gif.

Add database code to index.php

Edit your index.php to add the following statement verbatim after the comment. Look at what the code does, too.

```
<!-- add mysql connection here -->

<?php

$conn =   new mysqli($_SERVER['RDS_HOSTNAME'], $_SERVER['RDS_USERNAME'],
$_SERVER['RDS_PASSWORD'], 'ebdb');

// Check connection
if (mysqli_connect_errno()) {
     echo "Failed to connect to MySQL: " . mysqli_connect_error();
     http_response_code(500);
     exit();
}

$random = rand(0, 100000);
$user = "'user" . $random . "'";
$statement = $conn->prepare("INSERT into user (id, username) VALUES (?, ?)");
$statement->bind_param('is', $random, $user);
$statement->execute();

mysqli_close($conn);
echo "Cloud Computing Auto Scaling<br>Engineered for Heavy Load: ";
echo $random;

?>
```

You need to edit the mysql connection to match your MariaDB hostname, username and password, similar to this example:

```
$conn =   new mysqli('autoscaling.cekjl3c7gsvo.us-west-2.rds.amazonaws.com',
'root','123456', 'ebdb');
```

C. Set up PHP Web server

Configuring the Linux Server

Get a new Amazon Linux AMI instance (micro instance free tier) that you will use as your (PHP) web server. Use the same AZ as your previous instances.

SSH into your instance using your .pem file

PHP

Install php.

```
$ sudo yum install php-mysqli php php-xml php-mbstring php-cli mariadb105 httpd
tcpdump emacs
```

Press ‘y’ for each of the prompts that shows up. Note that you’re logged in as ec2-user, so you need to *sudo* all of these commands.

Web Server

First, let's create a test PHP file that will be accessed by the browser. Create directories so that you can put your file in `/var/www/html`.

Copy the 3 files for the web app that you have edited to function correctly to `/var/www/html` (using cyberduck, scp, putty etc.).

Restart (or start) `httpd`:

```
$ sudo systemctl start httpd
```

You can verify that httpd is on by running the following command:

```
$ sudo systemctl is-enabled httpd
```

Make httpd auto-start when boot.

```
$ sudo systemctl enable httpd
```

Security Groups

In the *AWS Management Console*, click on "Security Groups" in the Navigation panel. Select the *Security Group* that you used for the instance (probably the default one). Under the "Inbound" tab, add an HTTP rule (port 80). Click "Apply Rule Changes", and you should now be able to access your website!

Test that it works

In your browser, go to `http://ec2-hostname/index.php`

Check your table in MariaDB (using command-line client, MySQL workbench or other database client) and the webpage to see if your database code works. If your web server cannot connect to your database server, you will see an error on the webpage. Make sure you open up the right ports for the "Security Groups" for your MariaDB instance.

Run siege against your web application

```
$ siege -c10 -d0.5 -r10 http://webserverhostname/index.php
```

Look at the results from siege. Look at the monitoring panel on your console to see if your EC2 and RDS instances are running out of resources.

D. Experiment with siege to generate load

Use siege from your EC2 instance to trigger overload, adjusting the values for the flags (-c, -d, -r) to vary the load.

```
$ siege -c10 -d0.5 -r10 http://webserverhostname/index.php
```

Look at the monitoring panel for your 3 instances to see how load changes. What is the maximum load you can trigger with siege before you start to see HTTP status errors (not HTTP 200 OK)?

E. Write your lab report

1. Understanding overload
1.1 What option did you use to run siege in order to trigger overload on your web servers and get HTTP status errors or the server becomes unresponsive.
1.2 For the experiment using the siege options in 1.1, what is the average response time, transaction rate, and concurrency? How many of your transactions were successful vs. failed?

2. Take screenshots from the monitoring panel for your 3 instances to show load changes and point out when you start to HTTP status errors (not HTTP 200 OK) or when the server becomes unresponsive.

3. Do you think Amazon is providing sufficient resources for this instance size, price point, and number of clients you would generally have for a web application?

4. Where is the bottleneck in this system: your client, your web server, or your database server? Why do you think so? Provide evidence.

F. FAQ

Make sure you understand the commands before you run them.

1. How to allow remote connections to MariaDB (make sure you understand the commands before you run them)

```
# allow remote connections
echo "[mysqld]" | sudo tee -a /etc/mysql/my.cnf
echo "bind-address = 0.0.0.0" | sudo tee -a /etc/mysql/my.cnf
sudo systemctl restart mariadb
```

2. How to create database uses and grant privileges

```
# create db user
sudo mysql << EOF
CREATE DATABASE ${database_name};
CREATE USER ${database_user}@'%' IDENTIFIED BY '${database_pass}';
GRANT ALL ON ${database_name}.* TO ${database_user}@'%'; FLUSH PRIVILEGES; EOF
```

3. Hints on command-lie options for siege. Try ramping up so that you can understand how response time changes as a function of load. Listen to the instructor's suggestions in class on repetitions and delay.